Universität Tübingen
Rechnernetze und Internet
Prof. Dr. Georg Carle

Optimierung von virtuellen privaten Netzen mit Peer-2-Peer-Technologien

Studienarbeit

Lehrstuhl für Rechnernetze und Internet
Wilhelm-Schickard-Institut für Informatik
Fakultät für Informations- und Kognitionswissenschaften
Universität Tübingen

von

cand. inform.
Christian Korscheck

Betreuer:
Prof. Dr.-Ing. Georg Carle
Dipl.-Ing. Dirk Haage
Dipl.-Inform. Ralph Holz

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Tübingen, den 28. Juli 2008

Kurzfassung:

Die Studienarbeit erörtert die Anforderungen an ein Virtuelles Privates Netzwerk und untersucht anhand existierender Softwarelösungen die Stärken und Schwächen herkömmlicher Ansätze, die auf eine Client-/Server-Architektur setzen. Es wird das Konzept einer VPN-Software vorgestellt, das diesen Schwächen begegnen soll, indem es auf Peer-to-Peer-Technologien setzt. Anhand dieses Konzepts wurde eine Beispielimplementierung entwickelt, die ebenfalls vorgestellt wird.

Anschließend werden die Ergebnisse eines Vergleichstests zwischen einem bekannten Vertreter klassischer VPN-Software und der Beispielimplementierung im Rahmen dieser Studienarbeit betrachtet. Der direkte Vergleich der unterschiedlichen Herangehensweisen gibt Aufschluss über deren bevorzugte Einsatzgebiete. Daraus lässt sich ableiten, in welche Richtung sich Virtuelle Private Netzwerke in Zukunft entwickeln können.

Inhaltsverzeichnis

1. Einleitung

In der heutigen Zeit spielt die Vernetzung von Systemen eine wichtige Rolle. In der Firmenwelt benötigen Außendienstmitarbeiter nicht nur innerhalb, sondern auch außerhalb der Firma Zugriff auf das Firmennetzwerk. Firmenstandorte in verschiedenen Ländern müssen aus Gründen der Produktivität und des Verwaltungsaufwands miteinander verbunden werden, sodass sie Teil eines einzigen großen Firmennetzwerks werden. Lehreinrichtungen, wie Schulen oder Universitäten, aber auch öffentliche Plätze oder Gebäude, Privathaushalte oder Gruppen von Privatanwendern, die über die ganze Welt verteilt sind, profitieren von einem privaten Netzwerk, welches den Datenverkehr nach außen hin abschirmt und Zugriff auf die Daten im Netzwerk nur bestimmten Personengruppen ermöglicht. Die Gründe für die Vernetzung von Systemen sind zahlreich und können höchst unterschiedlich sein, doch ein Bedürfnis haben die meisten Vernetzungsvorhaben gemeinsam: Den Wunsch nach Privatsphäre und Sicherheit.

Eine Lösung Privatsphäre und Sicherheit in Vernetzungen herzustellen, stellen *Virtuelle Private Netzwerke* (VPNs) dar. Sie bieten Sicherheit durch Verschlüsselung und Wahrung der Integrität des Datenverkehrs und Privatsphäre durch strenge Authentifikations- und Autorisierungsfunktionen der beteiligten Systeme.

Aktuelle VPN-Softwarelösungen basieren primär auf einer Client-/Server-Architektur. Dabei stellt der Server die zentrale Einheit dar, mit dem sich die Clients verbinden und über den der Datenverkehr läuft. Durch diesen zentralen Ansatz stellt der VPN-Server einen *Single Point of Failure* dar. Fällt er aus, versiegt der Datenfluss zwischen allen Clients und das gesamte VPN bricht zusammen. Ein Serverausfall kann dabei in Folge eines Angriffs (*Denial-of-Service* beispielsweise) auftreten, aber auch in Folge einer zu hohen Belastung durch die Clients.

Die Durchsatzrate der Clients wird durch den Upstream des Servers begrenzt. Clients mit einem hohen Up- und Downstream können in einer solchen Netzwerkarchitektur nur dann eine optimale Performance entwickeln, wenn der Upstream des Servers so groß ist, dass er in der Lage ist den Datenstrom mehrerer, gleichzeitig sendender, Clients weiterzuleiten.
Die zunehmende Verbreitung von Breitbandinternetzugängen mit hohen Upstreams führt dazu, dass eine effiziente Auslastung in einem klassischen VPN mit einem einzelnen Server kaum noch ohne Einschränkungen möglich ist. Es müssen entweder Abstriche gemacht werden in Bezug auf die maximal erreichbare Durchsatzrate eines jeden Clients oder in Bezug auf die Anzahl der Clients, die gleichzeitig mit vollem Upstream Daten senden können. Wenn der Server die Clients in ihren Durchsatzraten beschränkt und die maximale Auslas-

tung vorhandener Ressourcen nicht mehr möglich ist, spricht man von einem *Flaschenhals* (Bottleneck).

Des Weiteren bedeutet die Bereitstellung eines VPN-Servers größeren Aufwand. Zum einen muss zusätzliche Infrastruktur organisiert werden, zum anderen muss der Server entsprechend konfiguriert werden.

1.1 Motivation

Um der aktuellen Entwicklung immer schnellerer Internetzugänge mit vergrößerten Upstream gerecht zu werden, soll diese Arbeit erörtern, inwiefern insbesondere dem Problem des Single-Point-of-Failure und dem Problem des Flaschenhalses begegnet werden kann. Der hier verfolgte Ansatz zur Lösung der genannten Probleme ist der Einsatz eines Peer-to-Peer-basierten VPNs, um von der klassischen Client-/Server-Architektur wegzukommen. Die Peer-to-Peer-Topologie basiert auf direkten Verbindungen zwischen den einzelnen Teilnehmern. Der Datenverkehr fließt nicht mehr über eine zentrale Stelle, sondern direkt zwischen den beiden Kommunikationspartnern. Es gibt keinen zentralen Server, der speziell konfiguriert werden muss. Einer der Peers eröffnet das private Netzwerk, andere Peers treten dem Netzwerk bei. Dadurch wird das VPN mobil und dynamisch und verfügbare Netzwerkressourcen können besser ausgenutzt werden.

1.2 Zielsetzung der Arbeit

Das Ziel der Studienarbeit ist die Vorstellung einer Implementierung eines Peer-to-Peer basierten VPNs. Direkte Verbindungen zwischen den Kommunikationspartnern sollen den Austausch großer Datenmengen performant ermöglichen. Der Anspruch an Privatsphäre und Sicherheit soll durch verschlüsselte Kommunikationskanäle und eine einfache Authentifikation erfüllt werden. Anhand dieser Implementierung soll untersucht werden, inwiefern Peer-to-Peer-basierte VPNs eine Alternative zu klassischen VPNs auf Client-/Server-Basis darstellen.

1.2.1 Randbedingungen

Die Software stellt ein proof-of-concept dar, keineswegs eine Lösung für den Produktiveinsatz. Es wird allerdings ein Ausblick gegeben in Bezug auf Verbesserungsmöglichkeiten und Ideen zur Performancesteigerung.

Die Sicherheitsaspekte werden angerissen, können aber nicht in der Tiefe behandelt werden, wie sie für ein Produktivsystem vonnöten sind. Nach [Boyd93] muss es eine dritte Partei in Form einer Zertifizierungsstelle (*Certificate Authority*) geben, die eine Basis für eine sichere Kommunikation ermöglicht. Im Rahmen dieser Arbeit wird vorausgesetzt, dass jeder Peer bereits beim Beitritt über ein signiertes Zertifikat einer CA verfügt. Zertifikatserstellung und -austausch über das VPN sind nicht vorgesehen.

Außerdem werden NAT-Traversal und die Komplikationen durch den Einsatz von Firewalls nicht berücksichtigt.

Andere Sicherheitskonzepte, wie das *Web of Trust* sind ebenfalls denkbar, werden im Rahmen dieser Studienarbeit allerdings nicht weiter betrachtet.

1.3 Gliederung

Im Folgenden sollen zunächst grundlegende Begrifflichkeiten erklärt werden. Anschließend werden einige wichtige Vertreter der VPN-Software vorgestellt, mitsamt deren Vor- und Nachteilen und einer Abgrenzung zur eigenen Arbeit. Der Konzept-Teil behandelt die Anforderungen an das Peer-to-Peer-basierte VPN und gibt Ideen zu Lösungsansätzen. Im Kapitel „Implementierung“ wird gezeigt, wie die Lösungsansätze konkret umgesetzt wurden. Abschließend werden die Resultate aus den eingehenden Überlegungen vorgestellt und Verbesserungsmöglichkeiten und Ideen, in welche Richtung das System weiterentwickelt werden könnte, geboten.

2. Begriffe

Bevor ein Überblick über den Aufbau des eigentlichen VPNs gegeben wird, sollen zunächst einige Begrifflichkeiten in aller Kürze erläutert werden.

2.1 Client-/Server-Architektur

Die klassische Client-/Server-Architektur zeichnet sich durch ihre Zentralisierung aus (Abb. 2.1). Einziger Diensteanbieter ist der Server. Die Clientsysteme nehmen den Dienst, den der Server bereitstellt, in Anspruch und haben in der Regel keine direkte Verbindung zu den anderen Teilnehmern. Der gesamte Datenverkehr läuft über den Server.

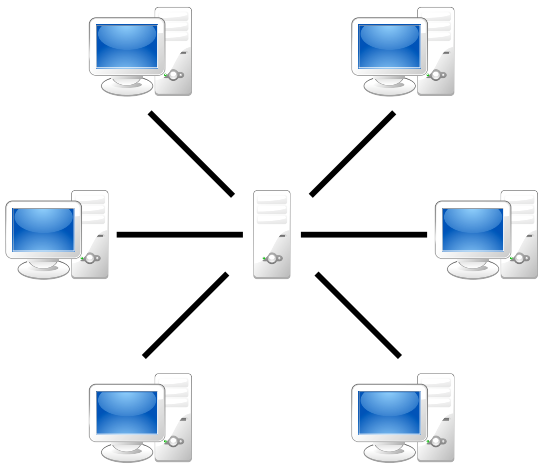

Abbildung 2.1: „Client-/Server-Architektur“

2.2 Peer-to-Peer-Netze

Ein Peer-to-Peer-Netz ist ein Zusammenschluss von Rechnern (Peers), bei dem alle Peers gleichberechtigt sind. Jeder Peer ist mit mindestens einem weiteren Peer auf direkte Weise verbunden; er kann aber auch mit einer Vielzahl anderer Peers verbunden sein.

Sind alle Peers mit jeweils allen anderen Peers verbunden, spricht man von einer *Vollvermaschung* (Abb. 2.2).

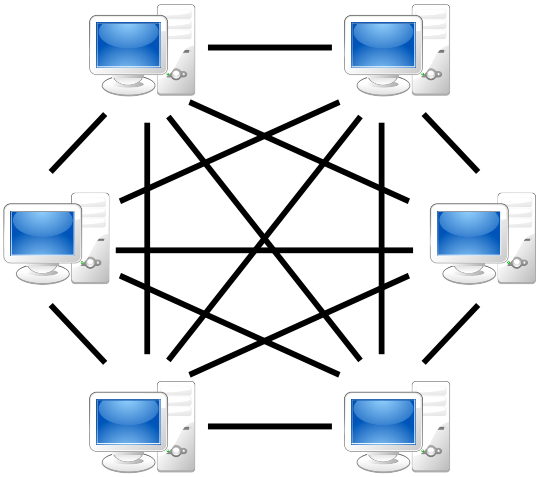

Abbildung 2.2: „Peer-to-Peer-Architektur (Vollvermaschung)“

Eine zentrale Instanz, wie beispielsweise einen Server, gibt es in modernen Peer-to-Peer-Systemen nicht. Zentralität ist allerdings noch in einigen Ansätzen zu finden. Das *hybride Konzept* weist sogenannte *Super-Peers* auf, zu denen sich die einzelnen Clients verbinden können. Die Super-Peers sind zuständig für das Routing innerhalb des Netzweks und damit privilegierter als „normale“ Peers ([Wehr05]).

Die Peer-to-Peer-Architektur selbst bietet nur wenig Komfort. In der Regel muss jeder Teilnehmer genau wissen, welche Dienste er von einem anderen Teilnehmer in Anspruch nehmen will und wie er diesen finden kann. Um diesem Problem zu begegnen, gehört zu einer modernen Peer-to-Peer-Architektur ein Overlay, welches insbesondere die Standardoperation *Lookup* implementiert. Die Lookup-Operation kann dazu verwendet werden denjenigen Peer zu identifizieren, der für eine bestimmte Objektkennung (im Folgenden *Key* genannt) zuständig ist. Dieser Key ist ein Hashwert, beispielsweise ein MD5-Hash einer IP-Adresse oder eines Dateinamens.

Ein naiver Ansatz, um ein bestimmtes Objekt in einem Peer-to-Peer-Netz zu finden, sieht so aus, dass der suchende Peer eine Anfrage an alle seine Nachbarn sendet und diese die Anfrage weiterleiten. So wird das gesamte Netzwerk bei der Suche nach einem bestimmten Objekt geflutet. Dabei kann es passieren, dass einige Pakete bestimmte Peers mehrfach erreichen. Eine *TTL* (Time-To-Live), die dekrementiert wird, sorgt dafür, dass die Pakete nicht endlos durch das Netzwerk geschickt werden. Dabei kann der Fall eintreten, dass eine entsprechende Anfrage niemals vom zuständigen Peer beantwortet wird, weil die TTL vorher auf 0 gesunken ist. Generell stellt sich das Problem, dass ein solches „strukturloses“ Netzwerk sehr schlecht skaliert. Bei einer bestimmten Anzahl von Teilnehmern machen die Suchanfragen einen gehörigen Teil des Traffics aus ([Wehr05]).

2.3 Distributed Hash Table und strukturierte P2P-Netze

Im Zusammenhang mit Peer-to-Peer-Netzen ist oftmals die Rede von verteilten Hashtabellen (Distributed Hash Table, DHT). Eine DHT ist ein Konzept zur gleichmäßigen und effizienten Verteilung von Routinginformationen über die Peers. Ziel ist es Suchanfragen zu beschleunigen, das Netzwerk zu stabilisieren und eine gute Skalierbarkeit zu erreichen ([Wehr05]).

Die Grundidee ist Folgende: Jeder Knoten verwaltet eine Routingtabelle mit einer bestimmten Anzahl von Einträgen für andere Knoten. In dieser Routingtabelle werden die

Keys den entsprechenden IP-Adressen im darunterliegenden (physischen) Netzwerk zugeordnet. Durch diese Verteilung der Routinginformationen auf alle Peers des Netzwerks sind alle Peers gleichberechtigt. Eine Anfrage wird an den Knoten weitergeleitet, der der entsprechenden Objektkennung am nächsten kommt. Eine Lookup-Operation kann dadurch in $O(\log n)$ Schritten durchgeführt werden. Die Optimierung der Lookup-Operation wird auf Kosten des Verwaltungsaufwandes der Routingtabellen erzielt. So müssen die Routingtabellen mehrerer Peers beim Beitritt oder beim Verlassen eines Knotens angepasst werden. Der Verwaltungsaufwand eines jeden einzelnen Peers beim Einsatz einer DHT ist damit zwar höher als in einem unstrukturierten Peer-to-Peer-Netzwerk, allerdings fällt dieser durch die deutlich besseren Kosten der Lookup-Operation und die gute Skalierbarkeit kaum ins Gewicht.

Aufbauend auf der Verteilung von Routinginformationen ist von einem *strukturierten* Peer-to-Peer-Netz die Rede, wenn jeder Key von mindestens einem Peer abgedeckt wird, sodass der gesamte mögliche Keyraum von jeweils einem Peer verwaltet wird. In der Regel deckt ein Peer aber nicht nur einen einzigen Key, sondern einen ganzen Teilbereich des gesamten Keyraums ab. In einem Netzwerk mit vier Peers ist Peer A beispielsweise zuständig für alle Keys von 00..00 bis 3F..FF, Peer B für alle Keys von 40..00 bis 7F..FF, Peer C für die Keys von 80..00 bis BF..FF und Peer D für die restlichen Keys von C0..00 bis FF..FF.

Der Routingmechanismus, der die Möglichkeit bietet Peers anhand eines Keys aufzufinden, wird *Key-Based-Routing* (KBR) genannt. Eine Nachricht für den Key AC..CC fällt im obigen Beispiel also in den Zuständigkeitsbereich von Peer C.

Die Abgrenzung zwischen DHT und strukturiertem Peer-to-Peer-Netz ist nicht ganz eindeutig, weswegen die beiden Begrifflichkeiten oftmals synonym verwendet werden.

2.4 SSL/TLS

SSL steht für Secure Sockets Layer und ist ein Verschlüsselungsprotokoll zur Datenübertragung im Internet ([DiAl99])[1]. *TLS* (Transport Layer Security) ist die standardisierte Weiterentwicklung von SSL 3.0 (TLS 1.0 steht neu für SSL 3.1). Aufgrund des Bekanntheitsgrades wird in dieser Arbeit die Bezeichnung SSL verwendet. Wie der Name bereits vermuten lässt, befindet sich SSL auf der Transportebene unmittelbar über dem TCP-Protokoll. Es setzt auf TCP/IP auf und besteht aus den beiden Komponenten *SSL Record Protocol* und *SSL Handshake Protocol*.

Erstere Komponente dient zur Verschlüsselung der Daten und zur Wahrung der Integrität der Nachrichten. Als Verschlüsselungsalgorithmen kommen symmetrische Algorithmen, wie DES, 3DES oder AES zum Einsatz. Die Nachrichten-Integrität erfolgt über die Bildung kryptografischer Prüfsummen, u. a. durch den SHA-1- und den MD5-Algorithmus.

Die Komponente „SSL Handshake Protocol“ dient dem Verbindungsaufbau, indem sich die Kommunikationspartner identifizieren und authentifizieren müssen und der kryptografische Algorithmus für die spätere Verschlüsselung der Daten ausgehandelt wird. Die Authentifizierung läuft in der Regel zertifikatsbasiert mit Hilfe asymmetrischer Algorithmen ab und baut damit auf der Public-Key-Kryptografie auf.

2.5 user space und kernel space

Als *user space* bezeichnet man den Bereich im Arbeitsspeicher, in dem Anwendungen operieren können. Im *kernel space*, ein für den Kernel reservierter Bereich im Arbeitsspeicher,

[1] Der SSL-Standard wird im RFC 2246 festgelegt. TLS wurde durch eine Reihe von RFCs erweitert, die unter http://www.ietf.org/ zu finden sind.

befinden sich u. a. hardwarenahe Treiber, die nur über *system calls* angesprochen werden können.

Diese Unterscheidung ist notwendig, um Verwaltungsaufgaben des Betriebssystems vor unerlaubtem Zugriff zu schützen.

2.6 TUN-Device

TUN ist ein virtueller Netzwerk-Kerneltreiber, der Netzwerkgeräte über Software simuliert. Hinter einem Netzwerkgerät (wie beispielsweise *eth0*) verbirgt sich normalerweise eine entsprechende Hardware in Form einer Netzwerkkarte. Die Hardware wird über einen Kerneltreiber angesprochen. Die Pakete werden in Form von elektrischen Impulsen über eine physische Leitung versendet. Im Gegensatz dazu werden die Pakete, die an ein TUN-Device gesendet werden, nicht auf eine physische Leitung geschickt, sondern an ein Programm im user space weitergeleitet. Die Pakete im user space können anschließend von der Anwendung gelesen werden.
Aus Sicht der Anwendung stellt das TUN-Device eine IP-Schnittstelle dar. Die Anwendung funktioniert wie gehabt, unabhängig davon, woher die Pakete kommen.
Eine Anwendung kann nicht nur Pakete lesen, sondern auch auf ein TUN-Device schreiben. Je nach Konfiguration werden die Pakete durch das Betriebssystem entsprechend weitergeleitet und können von anderen Anwendungen weiterverarbeitet werden.

In Zusammenhang mit TUN wird oftmals auch TAP erwähnt (TUN/TAP). Dabei handelt es sich um einen Treiber, der ein Ethernet-Gerät simuliert. Im Gegensatz zum TUN-Device, das sich auf Layer 3 des OSI-Modells befindet, ist das TAP-Device auf Layer 2 angesiedelt. TAP-Devices finden in dieser Arbeit allerdings keine Verwendung.

3. Related Work

Es gibt eine Vielzahl von VPN-Software. Die wichtigsten Lösungen sollen im Folgenden vorgestellt werden. In der Regel handelt es sich um Software, basierend auf der Client-/Server-Architektur. Dennoch existieren bereits auch Lösungen, bei denen sich in Ansätzen eine Peer-to-Peer-Architektur erkennen lässt.

3.1 IPsec

IPsec war eine der ersten Bemühungen sichere Verbindungen auf IP-Ebene (Layer 3) aufzubauen. Die erste Version wurde bereits 1998 entwickelt ([KeAt98])[1]. Es sollte die Schwächen des Internetprotokolls (IP) beheben. Das Hauptziel stellt die Gewährleistung von Vertraulichkeit, Authentizität und Integrität dar.

Im Rahmen der langjährigen Entwicklung und kontinuierlichen Verbesserung nahm auch die Komplexität zu, was von Kritikern ([ScFe99]) besonders bemängelt wird. IPsec gilt allerdings ab einer gewissen Schlüssellänge als sicher. IPsec verwendet das Schlüsselaustauschprotokoll IKE und verwaltet die Schlüssel automatisch. Es werden sowohl symmetrische, als auch asymmetrische Schlüssel verwendet. Welche Art von Schlüssel eingesetzt werden soll, wird beim Verbindungsaufbau zwischen den beiden Teilnehmern ausgehandelt. IPsec verändert den IP-Stack im *kernel space* und erfordert für jedes Betriebssystem eine eigenständige Implementierung.

Durch das Öffnen von sicheren Verbindungen zu anderen Knoten lässt sich mit Hilfe von IPsec ein dezentrales privates Netzwerk aufbauen. Trotz der gebotenen Sicherheit findet IPsec in dieser Studienarbeit keine Verwendung. Dies liegt insbesondere daran, dass die hohe Komplexität die Implementierung erschwert. Außerdem verfolgt das Konzept dieser Arbeit einen anderen Ansatz (SSL), wie später noch erläutert wird.

Es gibt allerdings VPN-Software, die auf IPsec setzt. Zu nennen ist an dieser Stelle der VPN-Client von Cisco.

[1] Das Hauptdokument zur Beschreibung von IPsec ist das RFC 2401. Weitere relevante RFCs sind unter http://www.ietf.org/ zu finden.

3.2 OpenVPN

OpenVPN[2] ist eine der bekanntesten Softwarelösung zum Aufbau eines VPNs. Dies rührt daher, dass OpenVPN OpenSource und damit kostenlos und für alle gängigen Betriebssysteme verfügbar ist. Es wird bereits seit einigen Jahren kontinuierlich weiterentwickelt. Aktuell ist die Version 2.1. Im Laufe der Zeit wurden eine Vielzahl von Betriebsmodi und Features entwickelt, die OpenVPN zu einem mächtigen Werkzeug machen.

Obwohl OpenVPN aufgrund der klassischen Client-/Server-Architektur die in der Einleitung aufgeführten Probleme mit sich bringt (Bottleneck, Single Point of Failure), soll die Funktionsweise der Software kurz erläutert werden, denn OpenVPN ist einer der wichtigsten Vertreter von VPN-Software.

OpenVPN kennt die beiden Betriebsmodi *Routing* und *Bridging*. Im Routing-Modus wird ein verschlüsselter Tunnel zwischen zwei Gegenstellen (Client und Server) hergestellt, über den die Pakete geleitet werden (Layer 3 bzw. 4). Jedem Kommunikationspartner wird eine virtuelle IP-Adresse in einem fiktiven Subnetz zugewiesen. Die VPN-Software kümmert sich dann um die Auflösung dieser virtuellen IP-Adresse in die reale IP-Adresse. Die Kommunikation wird wahlweise über das UDP- oder das TCP-Protokoll abgewickelt.

Der Bridging-Modus setzt auf Layer 2 des OSI-Modells an. Hier ist das Tunneln von Ethernet-Frames möglich und es können alternative Protokolle eingesetzt werden (beispielsweise IPX oder AppleTalk). Außerdem können Broadcasts über das Netzwerk geschickt werden, was beispielsweise den Einsatz von Windows NetBIOS ermöglicht, das auf Broadcasts angewiesen ist. Durch diese beiden Betriebsmodi wird das Einsatzspektrum von OpenVPN deutlich erhöht.

Die Verschlüsselung erfolgt mit SSL. Das IPsec-Protokoll wird nicht unterstützt. OpenVPN kennt zum Schlüsselaustausch die Modi „Preshared-Keys“, zertifikatsbasierter Schlüsselaustausch und eine einfache Lösung mit Benutzername und Passwort.

Die virtuelle IP-Adresse und die entsprechende Translation im Routing-Modus wird durch die Einrichtung eines TUN-Devices realisiert. Im Bridging-Modus wird analog ein TAP-Device eingesetzt. Die Pakete einer Legacy-Anwendung werden von OpenVPN dann im user space weiterverarbeitet (daher auch der Untertitel „user space VPN“). Diese Funktionsweise wird auch im Peer-to-Peer-VPN eingesetzt und im Kapitel „Konzept“ genauer erläutert.

Im Kapitel „Evaluation“ wird ein direkter Vergleich zwischen OpenVPN und der im Rahmen dieser Studienarbeit entwickelten Software gezogen.

3.3 socat

socat[3] ist ein Commandline-Tool zur Einrichtung von bidirektionalen Verbindungen zwischen zwei *Channels*. Es stellt eine Weiterentwicklung des Programms netcat dar und verwendet sowohl das TCP- als auch das UDP-Protokoll. Es bietet SSL-Verschlüsselung und kann als (HTTP-)Proxy fungieren. Der Vorteil von socat ist, dass die Channels, die miteinander verbunden werden, keineswegs zwei Rechner sein müssen. Es kann sich auch um Pipes, Streams oder beispielsweise die stdin unter Linux handeln.

socat gilt als das „Schweizer Taschenmesser“ im Netzwerkbereich. Es ist mächtig, hat aber den Charakter eines Diagnosetools. Mit erheblichem Aufwand ist es prinzipiell möglich

[2] http://openvpn.net
[3] http://www.dest-unreach.org/socat/

durch geschicktes Öffnen von Verbindungen zwischen verschiedenen Rechnern ein dezentrales VPN mit Peer-to-Peer-Charakter einzurichten. Dabei ist jedoch keinerlei Komfort gegeben, insbesondere nicht der Komfort einer automatisch verwalteten Lookup-Table zur Auflösung von privaten IP-Adressen in öffentliche IP-Adressen. Die Lookup-Table muss für jeden Peer vollständig manuell eingegeben werden.

3.4 VPN-X

VPN-X[4] wird zum Teil als „Peer-to-Peer-VPN“ bezeichnet und soll deswegen in dieser Aufzählung nicht fehlen. Tatsächlich weist es allerdings eine klassische Client-/Server-Architektur auf. Peer-to-Peer-Verbindungen werden nur zum NAT-Traversal per UDP verwendet. Ein genauer Einblick ist wegen der kommerziellen Natur und des geschlossenen Quellcodes nicht möglich.

VPN-X wird von vielen Betriebssystemen unterstützt, da es in Java programmiert ist. Es setzt ebenfalls auf TCP und UDP. Zur Verschlüsselung wird die Java Secure Socket Extension (JSSE) verwendet.

3.5 X-Bone

X-Bone[5] wird seit 1997 am Information Sciences Institute an der University of Southern California, USA entwickelt. X-Bone ist weniger ein VPN, als vielmehr ein Multifunktionsoverlay für IP-Netzwerke. Mit Hilfe von X-Bone kann ein IP-Netzwerk über ein anderes IP-Netzwerk gelegt werden mit einer veränderten Topologie (Ring, Stern, ...). Diese Overlays werden VIs (Virtual Internet) genannt.

Es ist denkbar ein Overlay in Form einer Peer-to-Peer-Topologie zu gestalten. Die Verschlüsselung von Daten erfolgt durch IPsec bzw. SSL auf Zertifikatsbasis. Zur Datenübertragung werden das TCP- und das UDP-Protokoll eingesetzt. Mit Hilfe von X-Bone kann aufgrund dieser Eigenschaften ein VPN eingerichtet werden. Durch die hohe Komplexität und den großen Umfang stellt X-Bone allerdings keine komfortable (*lightweight*) Lösung dar.

3.6 Hamachi

Eine der bekanntesten und einem Peer-to-Peer-VPN am nächsten kommenden Möglichkeiten zur Errichtung eines VPNs stellt Hamachi[6] dar. Es ist nicht kommerziell, allerdings auch nicht OpenSource. Es bietet dem Benutzer die Möglichkeit eine Freundesliste zu verwalten und spontan mit ausgewählten Freunden ein VPN zu eröffnen. Dabei dient der Server des Anbieters als Lookup-Table für die öffentlichen IP-Adressen. Die einzelnen Rechner müssen zunächst die öffentliche IP-Adresse der Freunde nachsehen und öffnen anhand dieser IP einen Tunnel zum Zielsystem.

Fällt der Server aus, kann kein neuer Teilnehmer dem bereits bestehenden Netzwerk beitreten, weil die einzige Lookup-Möglichkeit verlorengegangen ist. Bereits geöffnete Tunnel (also bestehende Netzwerke) bleiben bestehen und können weiterhin genutzt werden. Der Datenverkehr läuft über den geöffneten Tunnel und nicht über den Server. Es wird also mit Hilfe eines zentralen Servers ein Peer-to-Peer-Netzwerk aufgebaut.

[4] http://birdssoft.com
[5] http://www.isi.edu/xbone/
[6] http://www.hamachi.cc

Hamachi findet vor allem Verwendung bei Computerspielen und Filesharing, nicht jedoch in Firmenumgebungen.

Der Datenverkehr wird zwar verschlüsselt, doch durch den geschlossenen Quelltext ist nicht gesichert, ob die Verschlüsselungsalgorithmen wirklich sicher sind. Die Authentifizierung läuft allerdings nicht über Zertifikate, sondern über die soziale Komponente. Ein privates Netzwerk kann nur mit Freunden eröffnet werden. Es muss anderweitig für Authentizität gesorgt werden.

4. Konzept

Dieses Kapitel stellt das zugrundeliegende Konzept der Peer-to-Peer-basierten VPN-Software vor, die im Rahmen dieser Studienarbeit implementiert wurde. Nach der Aufstellung der wichtigsten Anforderungen an eine VPN-Software im Allgemeinen werden die einzelnen Komponenten näher erläutert.

4.1 Anforderungen

Eine der Hauptaufgaben eines jeden VPNs ist der Aufbau eines Tunnels zu einem anderen System, um Pakete einer Legacy-Anwendung über diesen Tunnel zu schicken. Dabei spielt es zunächst keine Rolle, ob es sich beim Kommunikationspartner um einen privilegierteren Server oder um einen gleichberechtigten Peer handelt.

4.1.1 Grundlegende Anforderungen

4.1.1.1 Schnittstelle zur Legacy-Anwendung

Die Schnittstelle zur Legacy-Anwendung wird durch die Verwendung eines TUN-Devices realisiert. Die Verwendung eines TUN-Device bringt den Vorteil mit sich, dass es bereits funktionierende Implementierungen dieses Treibers für alle gängigen Betriebssysteme gibt. Das TUN-Device lässt sich wie eine Datei öffnen. Lese- und Schreiboperationen können problemlos darauf angewendet werden. Das Routing der Pakete im Zusammenhang mit dem TUN-Device wird durch das Betriebssystem anhand der entsprechenden Einträge in der betriebssysteminternen Routingtabelle vorgenommen. Die Routingtabelle wird beim Öffnen des TUN-Device um einen entsprechenden Eintrag erweitert.

Die Verwendung des TUN-Devices als Schnittstelle zur Legacy-Anwendung empfiehlt sich insbesondere aufgrund seiner einfachen Handhabung, da das Routing durch das Betriebssystem übernommen wird.

4.1.1.2 Tunnelaufbau

Der Aufbau eines Tunnels für den Datenverkehr zwischen zwei Knoten erfolgt über einen Socket. Sockets bilden eine plattformunabhängige, standardisierte Schnittstelle zwischen der Netzwerkprotokoll-Implementierung des Betriebssystems und der eigentlichen Anwendung. Durch die standardisierte API lassen sich Sockets auf den unterschiedlichsten Betriebssystemen auf die gleiche Weise ansteuern. Da auch Sockets wie Dateien geöffnet,

gelesen und beschrieben werden können, indem der entsprechende File-Deskriptor angesprochen wird, ist die Kommunikation über ein physisches Netzwerk anhand eines Sockets besonders einfach zu handhaben.

4.1.1.3 Routing

Die Schnittstelle zur Legacy-Anwendung und der Aufbau eines Tunnels bilden bereits eine funktionierende Grundlage für ein VPN. Die Routingtabelle des Betriebssystems wird durch das Öffnen des TUN-Devices angepasst. Pakete können über Sockets an entfernte Systeme gesendet werden. Es wird nun noch ein Verbindungsstück benötigt, welches die Pakete vom TUN-Device liest und auf einen Socket schreibt und anders herum. Dieses Verbindungsstück stellt die eigentliche VPN-Software dar (Abb. 4.1).

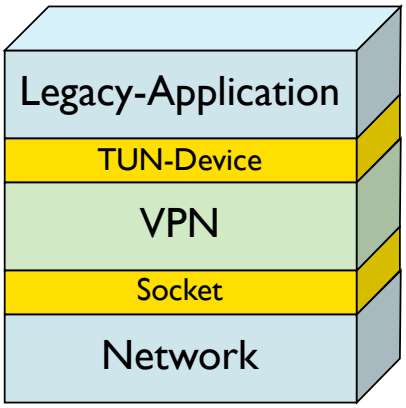

Abbildung 4.1: „VPN-Software und Schnittstellen“

4.1.2 Weitere Anforderungen

Die Hauptaufgaben eines VPNs wurden bereits erläutert. Ein Peer-to-Peer-gestütztes VPN muss allerdings noch weitere Anforderungen erfüllen.

4.1.2.1 Verbesserte Datenübertragungsgeschwindigkeiten

Das Hauptproblem eines klassischen VPNs, welches auf einer Client-/Server-Architektur aufbaut, nämlich der Server als Flaschenhals (Abb. 4.2) und Single Point of Failure, soll durch die Dezentralisierung mit Hilfe eines Peer-to-Peer-Netzes umgangen werden.

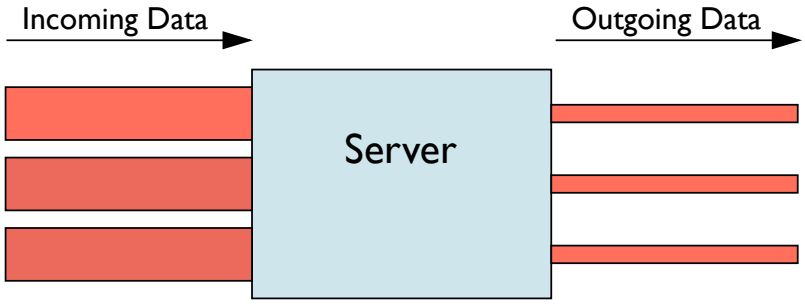

Abbildung 4.2: „Flaschenhals“

In einem klassischen VPN ist der Upstream eines jeden Peers die Obergrenze für die Geschwindigkeit bei der Datenübertragung. Ein Knoten, dessen Internetanbindung einen

maximalen Upstream von 1MBit/s erlaubt, wird keine Möglichkeit haben diesen Wert zu überschreiten. Ein schwacher Upstream eines Servers kann die einzelnen Knoten allerdings bremsen. In einem Szenario, bei dem ein Server über einen maximalen Upstream von 1MBit/s verfügt, jedoch die Daten von drei gleichzeitig sendenden Knoten mit ebenfalls je 1MBit/s Upstream weiterleiten muss, bremst der Server die Knoten, sodass jeder von ihnen effektiv nur noch mit etwa 330KBit/s senden kann.

Durch den Aufbau von direkten Tunneln zu den Kommunikationspartnern wird die Geschwindigkeit bei der Datenübertragung einzig und allein durch den Upstream des Peers bestimmt und keineswegs mehr durch einen schwachen Server beeinflusst. Sendet ein Peer mit einem maximalen Upstream von 1MBit/s an drei weitere Peers Daten, so ist sein Upstream pro Tunnel zwar effektiv auch auf etwa 330KBit/s reduziert, allerdings können die anderen Peers im VPN nach wie vor ihre 1MBit/s vollständig auslasten. Dies macht das Peer-to-Peer-gestützte VPN effizienter. Die Datenübertragungsgeschwindigkeiten können durch Dezentralisierung natürlich nur dann wirklich verbessert werden, wenn es nicht einen Server gibt, der einen um ein Vielfaches höheren Upstream besitzt als die Summe der Peers im Netzwerk. Hat ein Server beispielsweise einen Upstream von 100MBit/s und befinden sich im VPN 10 Peers mit einem Upstream von jeweils 1MBit/s, so wird keiner der Peers gebremst.

Wie bereits in der Einleitung kurz erwähnt, entsteht vor allem durch den Trend zu immer größeren Upstreams von Breitbandinternetzugängen die Gefahr, dass die Knoten ihre maximale Leistung nicht vollständig ausspielen können. Aktuell sind 1MBit/s Upstream keine Seltenheit mehr. Ein Server mit 100MBit/s Upstream kann also lediglich ein Netzwerk mit maximal 100 Clients ohne Einschränkungen bedienen.
(Anmerkung: In dieser Überlegung wird weder die Netzwerklast, die durch Verwaltungsaufgaben entsteht, noch die Auslastung von CPU/RAM des Servers berücksichtigt. Außerdem wird angenommen, dass jeder Client zu jedem Zeitpunkt seinen Upstream vollständig auslasten möchte, beispielsweise durch die Übertragung größerer Dateien.)

Die Dezentralisierung als Lösungsansatz wird dadurch erreicht, dass ein Peer nicht nur einen einzigen Tunnel, sondern eine Vielzahl von Tunneln zu verschiedenen Kommunikationspartner aufbauen kann.

Zusätzlich zum Wegfall einer Begrenzung durch den Upstream bieten direkte Tunnel den Vorteil, dass die Kommunikationswege kürzer werden. In einem klassischen VPN wird der gesamte Datenverkehr über den Server abgewickelt. Der Server ist in jedem Fall ein notwendiger Zwischenstopp und bedeutet eine erhöhte Latenzzeit. Hereinkommende Pakete müssen auf ihr Ziel (Destination-IP-Address) hin untersucht und an den richtigen Client gesendet werden. Durch den Einsatz eines direkten Tunnels ist der Endknoten immer direkt der Ziel-Peer. Es sind keine Zwischenstopps vonnöten.
(Anmerkung: Diese Überlegungen berücksichtigen nicht das physische Routing von Paketen im Internet, bei dem zwangsläufig mehrere Server oder Knotenpunkte passiert werden müssen oder auch ein „schlechter“ Weg gewählt werden kann. Das Routing der Pakete im Internet kann durch die VPN-Software nicht beeinflusst werden.)

4.1.2.2 Mehrere Tunnel

Die Schnittstelle zur Legacy-Anwendung benötigt in einem Peer-to-Peer-gestützten VPN keine zusätzlichen Anforderungen, sondern verhält sich genauso, wie in den Anforderungen an ein klassisches VPN; es wird auch in einem Peer-to-Peer-gestützten VPN nur ein einziges TUN-Device benötigt. Um mehrere Tunnel verwenden zu können, muss für jeden Tunnel ein Socket geöffnet werden.

Kommen Datenpakete über die Netzwerkschnittstelle herein, werden sie durch die VPN-Software zur Legacy-Anwendung „weitergeleitet“, indem sie auf das TUN-Device geschrieben werden.

Sollen die Datenpakete von der Legacy-Anwendung zum Zielhost geschickt werden, werden die Daten vom TUN-Device gelesen und anschließend auf den entsprechenden Socket, der die Verbindung zum Zielhost darstellt, geschrieben. Abbildung 4.5 verdeutlicht diesen Vorgang.
Das Routing der Pakete und die Verwaltung der Sockets sind zwei zentrale Aufgaben des Peer-to-Peer-gestützten VPNs.

4.1.2.3 Sicherheit

Keineswegs notwendig, aber durchaus wünschenswert ist die Verschlüsselung der Kommunikationsinhalte, um die Daten vor dem Zugriff durch Dritte zu schützen. Die Verschlüsselung per SSL/TLS ist ein möglicher Lösungsweg, dessen Vor- und Nachteile an dieser Stelle erörtert werden sollen.

Die Funktionsweise von SSL wurde bereits erwähnt. Da SSL auf der Transportebene angesiedelt ist, verwendet es ebenfalls Sockets zur Datenübertragung. SSL geht konform mit den im vorherigen Abschnitt diskutierten Anforderungen an ein VPN.
Die zertifikatsbasierte Authentifizierung (der SSL-Handshake) ist verhältnismäßig rechenintensiv. Es besteht allerdings keine Notwendigkeit für einen performanten Verbindungsaufbau im VPN. Idealerweise werden Tunnel zwischen zwei Peers über einen längeren Zeitraum aufrecht erhalten, sodass das SSL-Handshake nur sehr selten durchgeführt werden muss. Wichtig ist insbesondere die Performance bei der Datenübertragung. Die symmetrische Verschlüsselung nimmt, je nach verwendetem Algorithmus, nur noch wenig Rechenzeit in Anspruch und bietet sich für den Einsatz im VPN besonders an.

In der Regel wird SSL der Nachteil zugeschrieben, dass es nur die Kommunikation zwischen zwei Stationen verschlüsseln kann. In manchen Szenarien ist der Datenversand über mehrere Stationen wünschenswert. In diesem Fall müssten zwischen je zwei Stationen die Daten neu verschlüsselt werden. Da in unserem Peer-to-Peer-gestützten VPN ausschließlich direkte Tunnel zwischen zwei Peers zum Einsatz kommen (Ende-zu-Ende-Verbindungen), spielt dieser Nachteil keine Rolle.

Aus diesen Überlegungen heraus bildet SSL die ideale Lösung zur Verschlüsselung der Daten im Peer-to-Peer-gestützten VPN.

Ein wichtiger Aspekt eines VPNs im Produktiveinsatz ist die Art und Weise, wie Zertifikate ausgestellt und ausgetauscht werden. Im Rahmen dieser Arbeit wird jedoch vorausgesetzt, dass jeder Peer bereits ein gültiges Zertifikat besitzt.

4.1.2.4 IP-Translation

Die Legacy-Anwendung kennt nur den privaten IP-Adressraum des VPNs und kommuniziert mit anderen Systemen über deren private IP-Adressen, die ihnen durch das VPN zugewiesen wurden. Es wird vorausgesetzt, dass die Legacy-Anwendung weiß, mit welcher privaten IP-Adresse sie kommunizieren will. Eine Broad- oder Multicast-Funktionalität wird im Rahmen dieser Arbeit nicht benötigt.

Die private IP-Adresse ist rein virtuell (*virtuell* und *privat* werden in dieser Arbeit synonym verwendet). Sie kann nicht dazu verwendet werden das Zielsystem (physisch) anzusprechen. Hierfür ist dessen reale IP-Adresse (im Folgenden *öffentliche* IP-Adresse genannt) vonnöten, zu der der Kommunikationskanal aufgebaut wird. Das VPN muss also eine Möglichkeit bieten die privaten IP-Adressen in öffentliche IP-Adressen aufzulösen. Dies könnte mit

Hilfe einer einfachen Hashtabelle gelöst werden, bei der die privaten IP-Adressen auf die entsprechenden öffentlichen IP-Adressen abgebildet werden. Diese Hashtabelle könnte auf einem System im VPN zentral abgelegt werden. Anfragen nach der Adressauflösung müssten an dieses System gerichtet werden. Der Nachteil dieses Lösungsansatzes ist, dass man in gewissem Maße wieder zurück zu einer zentralen Client-/Server-Architektur gelangt.

Der beste Lösungsansatz ist der Einsatz eines strukturierten Peer-to-Peer-Overlays, welches ein simples Nachrichtensystem mit key-based-routing implementiert. Auf diese Weise kann mit Hilfe eines Keys (in diesem Fall: die private IP-Adresse) eine Nachricht an einen Peer geschickt werden, der auf diese Nachricht mit seiner öffentlichen IP-Adresse antwortet. In einem strukturierten Peer-to-Peer-Netz ist jeder Peer für einen Teil des Keyraums zuständig. Es wird immer der gesamte Keyraum abgedeckt. Existiert kein Peer zu der gewünschten IP-Adresse, sendet ein anderer Peer, der für diesen Keyraum zuständig ist, eine entsprechende Antwort. Ein Timeout wird dadurch überflüssig.

4.2 Hauptkomponenten

Nachdem die Anforderungen an das VPN erörtert wurden, sollen im Folgenden die Hauptkomponenten umrissen werden, die die wichtigsten Funktionen im VPN übernehmen.

4.2.1 Peer-to-Peer-Ovelay

Bevor ein Tunnel über einen Socket geöffnet werden kann, muss die öffentliche IP-Adresse des Kommunikationspartners ermittelt werden. Die Ausgangssituation ist Folgende: Ein beliebiger Peer kennt in der Regel nur die private IP-Adresse seines Kommunikationspartners. Damit sich mehrere Systeme überhaupt in einem VPN organisieren können, muss jedem Peer eine Art „Server“ bekannt sein.

In klassischen VPNs wird diese Rolle von einem realen Server eingenommen. In unserem Fall gibt es einen sogenannten *Bootstrap*-Peer, der das VPN eröffnet und folglich als Einstiegspunkt für die anderen Teilnehmer dient. Mit der Eröffnung des VPNs wird das Peer-to-Peer-Overlay initiiert. Es dient in erster Linie dazu Peers in einer organisierten Struktur unterzubringen, um Nachrichtenaustausch zwischen den Peers zu ermöglichen. Der Nachrichtenaustausch beschränkt sich auf das Erfragen der öffentlichen IP-Adresse der anderen Teilnehmer anhand eines bestimmten Keys. Wie im Kapitel „Implementierung“ noch erläutert wird, wird der Key aus dem Hashwert der privaten IP-Adresse gebildet.

Ist der Teilnehmer erfolgreich beigetreten, kann er ab sofort die öffentlichen IP-Adressen der anderen Teilnehmer abfragen. Möchte er beispielsweise eine Verbindung zum Teilnehmer 10.0.0.7 aufbauen, so sendet er über das Peer-to-Peer-Overlay eine Nachricht an den Peer mit dem entsprechenden Key (dem Hashwert von 10.0.0.7). Als Antwort erhält er, sofern einer der Peers im VPN die private IP-Adresse 10.0.0.7 besitzt, die öffentliche IP-Adresse (Abb. 4.3). Existiert kein Peer mit der entsprechenden privaten IP-Adresse, bekommt er von einem anderen, für diesen Keyraum zuständigen Peer, eine entsprechende Fehlermeldung zugeschickt. Ein Tunnelaufbau ist dann nicht möglich. Fällt ein Peer aus ohne seine Daten (die Informationen über öffentliche IP-Adressen, die er speichert) an andere Peers abzugeben, gehen die Informationen verloren. Entsprechende Peers sind dann zunächst nicht erreichbar. Das korrekte Verlassen eines Peer-to-Peer-Netzes erhöht die Stabilität.

Verlief die Anfrage problemlos, kann der Teilnehmer ab sofort unabhängig vom Peer-to-Peer-Overlay einen Tunnel zum gewünschten Kommunikationspartner öffnen.

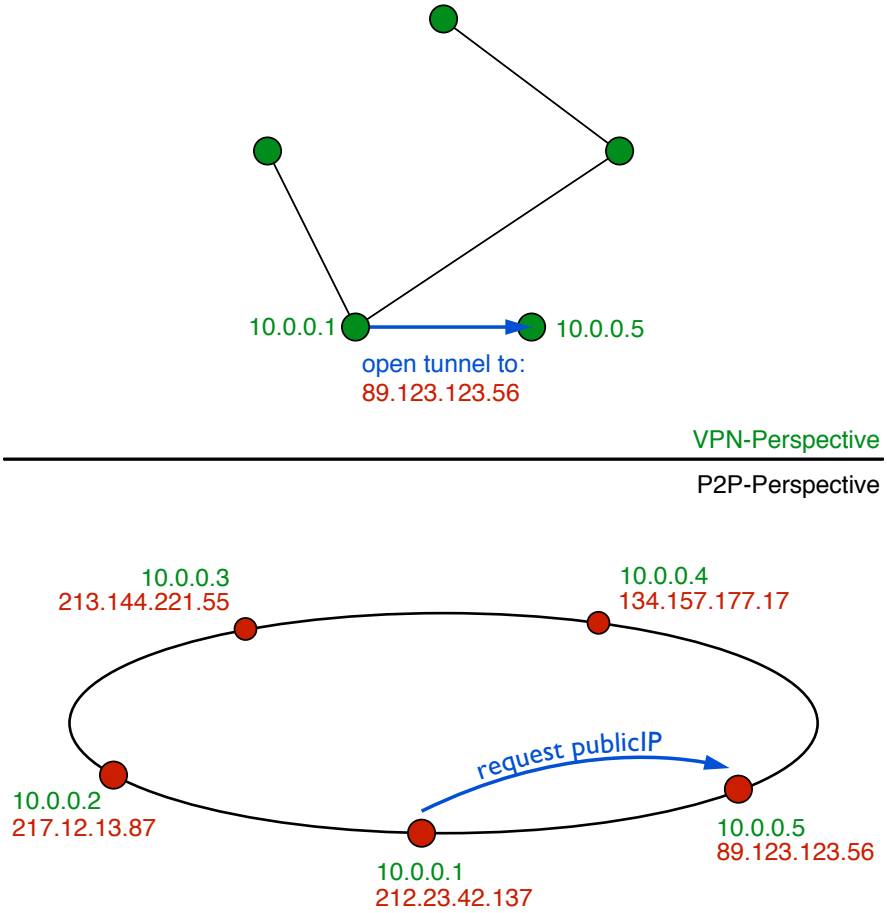

Abbildung 4.3: „Peer-to-Peer-Overlay“

4.2.2 Routing

Jeder Teilnehmer des VPNs verfügt über ein einziges TUN-Device, welches die Schnittstelle zwischen der VPN-Software und der Legacy-Anwendung darstellt. Dabei ist es irrelevant, welche Art von Paketen durch die Legacy-Anwendung erzeugt und verschickt werden. Es können TCP, UDP oder auch ICMP-Pakete sein, um nur einige Beispiele zu nennen. Wichtig ist lediglich, dass es sich auf Layer 3 um ein IP-Paket handelt.

Wenn ein Peer mit verschiedenen Teilnehmern parallel kommunizieren möchte, müssen die Pakete durch die VPN-Software auf den korrekten Tunnel geroutet werden. Dazu werden die IP-Header der Pakete, die vom TUN-Device gelesen werden, auf den *Destination Address*-Eintrag[1] überprüft. In diesem Feld steht die private IP-Adresse des Kommunikationspartners, wie sie im VPN verwendet wird.

Zur Verwaltung der geöffneten Tunnel dient eine Routingtabelle, die die private IP-Adresse des Kommunikationspartners auf den entsprechenden Socket-Filedeskriptor abbildet. Abbildung 4.4 zeigt zusätzlich eine Spalte mit der öffentlichen IP-Adresse des Kommunikationspartners, obwohl in der Routingtabelle eigentlich nur der Filedeskriptor abgelegt ist. Dies liegt daran, dass sich anhand des Filedeskriptors die öffentliche IP-Adresse problemlos auslesen lässt und diese in bestimmten Fällen zum Verbindungsaufbau benötigt wird. Nä-

[1] Der IP-Header wird im RFC 791 spezifiziert.

heres dazu wird im Abschnitt „Tunneling“ erläutert. Zur reinen Paketweiterleitung genügt jedoch der Socket-Filedeskriptor.

Kommt ein Paket über das TUN-Device herein, wird in der Routingtabelle des VPNs nachgesehen, ob bereits ein Tunnel zu diesem Kommunikationspartner geöffnet ist.

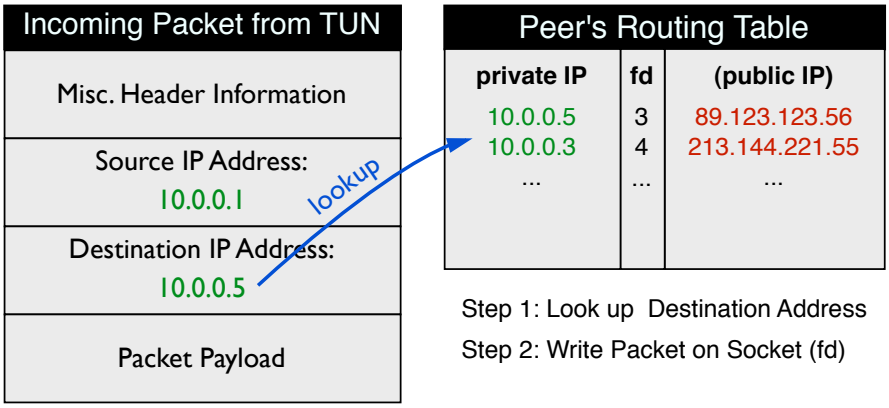

Abbildung 4.4: „Routingtabelle“

Existiert ein geöffneter Tunnel in der Routingtabelle, wird das Paket auf den entsprechenden Socket geschrieben. Es wird dadurch zum Payload eines neuen TCP/IP-Pakets (Abb.4.5).

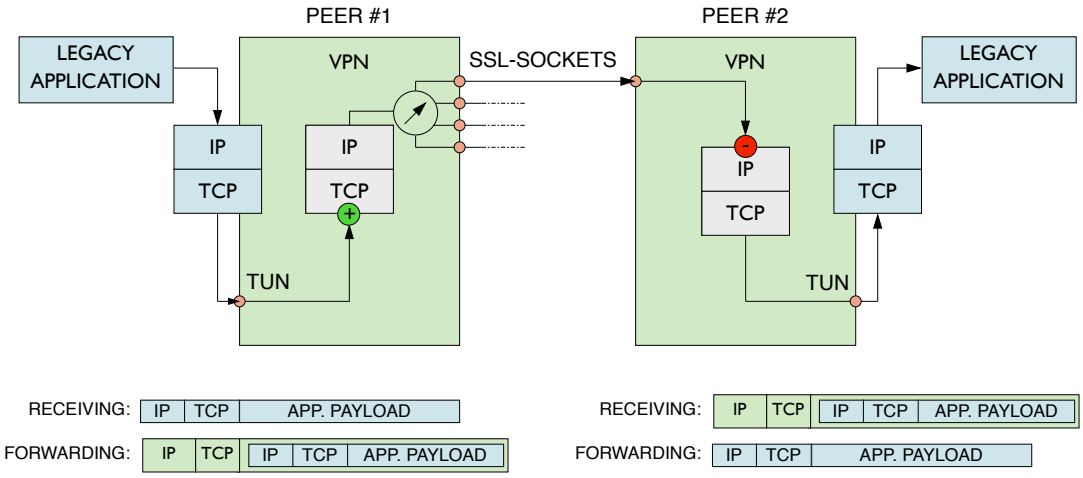

Abbildung 4.5: „Routing“

Befindet sich in der Routingtabelle kein entsprechender Eintrag, wird versucht ein Tunnel aufzubauen. Dieser Vorgang wird im Abschnitt „Tunneling“ erläutert.

Auf Empfängerseite kommt das Paket über einen Socket herein. Der Payload (das Paket der Legacy-Anwendung) wird gelesen und anschließend direkt auf das TUN-Device geschrieben. Das Betriebssystem leitet die Pakete vom TUN-Device weiter zur Legacy-Anwendung.

4.2.3 Tunneling

Die Routingtabelle wird immer dann um einen Eintrag erweitert, wenn ein Paket vom TUN-Device gelesen und auf den entsprechenden Socket geschrieben wird. Kommt bei Peer

A über das TUN-Device ein Paket herein, welches eine private Destination-IP-Address aufweist, die nicht in der Routingtabelle des VPN gelistet ist, müssen zwei Fälle unterschieden werden.

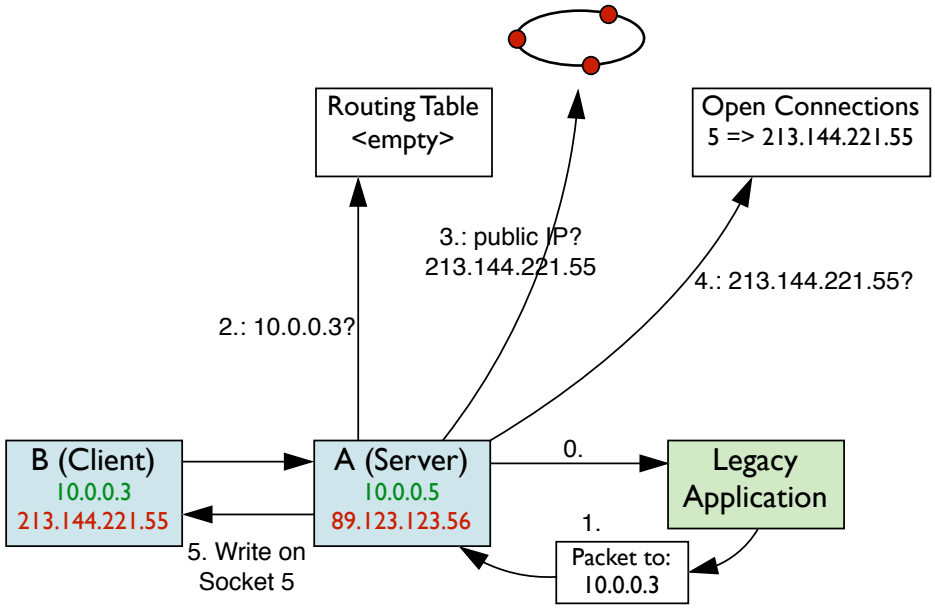

Abbildung 4.6: „Routing“

1. Fall (Abb. 4.6): A hat bereits Pakete von einem anderen Peer *B* angenommen, aber noch keine Pakete zurückgeschickt. A ist aus Sicht des TCP-Verbindungsaufbaus der *Server*, der an einem festen Port auf eingehende Verbindungen lauscht und diese gegebenenfalls animmt. B stellt den *Client* dar. Ein Paket, das über einen Socket hereinkommt, hat als Source-IP-Address nur die jeweilige öffentliche IP-Adresse von B, nicht jedoch die private IP-Adresse von B. Für einen Eintrag in die Routingtabelle wird allerdings die private IP-Adresse benötigt. Die Routingtabelle ist deswegen auf Peer A noch leer, obwohl bereits Datenpakete von B nach A geflossen sind (Schritt 0).
Nun soll ein Datenpaket von der Legacy-Anwendung von A nach B geschickt werden. Im Paket steht die private IP-Adresse von B. Peer A schaut in die eigene (noch leere) Routingtabelle, um die öffentliche IP-Adresse von B herauszufinden (Schritt 2). Da er B noch nicht kennt, fragt er das Peer-to-Peer-Overlay nach der öffentlichen IP-Adresse von 10.0.0.3 und erhält als Antwort 213.144.221.55. Nun könnte A eine Verbindung zu dieser IP-Adresse aufbauen. Da aber bereits Pakete von B nach A geflossen sind, muss es auch eine bereits geöffnete Verbindung geben. A sieht in einer weiteren Liste (der Socket-Liste) nach, ob zur IP-Adresse 213.144.221.55 bereits ein Socket existiert (Schritt 4). In dieser Socket-Liste stehen alle öffentlichen IP-Adressen, von denen bereits Pakete hereingekommen sind, denen allerdings noch die Zuordnung zur privaten IP-Adresse fehlen und deswegen noch nicht in der Routingtabelle abgespeichert sind.
Anhand der Socket-Liste stellt Peer A fest, dass bereits eine geöffnete Verbindung zu B über den Socket-Filedeskriptor 5 existiert. Er schreibt das Datenpaket auf diesen Socket (Schritt 5) und aktualisiert anschließend seine eigene Routingtabelle.

2. Fall: Es hat noch kein Datenverkehr stattgefunden. Die private IP-Adresse aus dem Feld Destination-IP-Address wird ausgelesen (10.0.0.3), um im Peer-to-Peer-Overlay die öffentliche IP-Adresse von B abzufragen, die dann wiederum dazu verwendet wird eine TCP-Verbindung aufzubauen. Im Gegensatz zum Szenario aus Fall 1, fungiert A aus Sicht

des TCP-Verbindungsaufbau als *Client* und B als *Server*. Es wird eine *connect*-Anfrage an B gesendet. Wird die Anfrage beantwortet und kann das SSL-Handshake erfolgreich abgeschlossen werden, ist der Tunnel geöffnet und die Pakete werden auf den entsprechenden Socket geschrieben. Außerdem wird die Routingtabelle von A sofort aktualisiert.

Kann bei der Durchführung des SSL-Handshakes ein Zertifikat nicht verifiziert werden, beispielsweise, weil es nicht von der gleichen CA signiert wurde oder weil es abgelaufen ist, wird der Verbindungsaufbau unterbrochen. Das Paket kann nicht versendet werden und wird verworfen.

4.2.4 Pakete

In Abbildung 4.5 wird der Weg eines Paketes von Peer 1 zu Peer 2 dargestellt. Dabei werden die Pakete einer Legacy-Anwendung in ein Paket, welches durch die VPN-Software erzeugt wird, verpackt.

Dieser Vorgang soll anhand des Programms *ping* erläutert werden. Peer 1 bekommt im VPN die virtuelle IP-Adresse 10.0.0.1 zugewiesen. Peer 2 besitzt die 10.0.0.5. Auf Peer 1 wird folgender Befehl aufgerufen.

```
> ping 10.0.0.5
```

Die Anwendung *ping* sendet nun ein ICMP-Paket los an das Zielsystem 10.0.0.5. Dieses Paket wird vom Betriebssystem geroutet. Dazu wird in der betriebssysteminternen Routingtabelle nachgesehen, dass das Paket auf das Gerät *tun0* geschrieben werden soll.
Nachdem das Paket auf *tun0* geschrieben wurde, wird das Paket durch den TUN-Treiber weitergeleitet und kann von der VPN-Software vom TUN-Device gelesen werden. Die VPN-Software bemerkt, dass am TUN-Device ein Paket anliegt, liest es, überprüft den IP-Header, prüft die eigene Routingtabelle und stellt fest, dass noch kein Tunnel nach außen hin zu 10.0.0.5 geöffnet wurde.

Peer 1 befragt das Peer-to-Peer-Overlay nach der öffentlichen IP-Adresse von 10.0.0.5 und bekommt als Antwort die öffentliche IP-Adresse 89.123.123.56, zu der er eine Anfrage zum Verbindungsaufbau stellt. Peer 2 bekommt die Anfrage zur Herstellung einer Verbindung von 213.144.221.55, akzeptiert sie und führt das SSL-Handshake durch. Der Tunnel ist geöffnet.

Peer 1 schreibt nun das ICMP-Paket auf diesen Socket. Dabei wird das ICMP-Paket als Payload in ein TCP-Paket verpackt.

Auf der Seite von Peer 2 kommt ein Paket über den geöffneten Tunnel herein. Der TCP/IP-Header wird wieder entfernt. Der Payload (das reine ICMP-Paket) wird von Peer 2 auf dessen TUN-Device geschrieben. Das Betriebssystem übernimmt die Weiterleitung des Paketes vom TUN-Device zur Legacy-Anwendung *ping*.

ping auf dem System von Peer 2 reagiert auf den soeben hereingekommenen *echo request* mit einem *echo reply* und sendet ein entsprechendes ICMP-Paket mit dem Zielsystem 10.0.0.1 los. Dieses nimmt wieder seinen Weg über das TUN-Device zur VPN-Software, wird dort auf den zugehörigen Tunnel geschrieben und dabei in ein TCP/IP-Paket gepackt und kommt bei Peer 1 an. Dort wird der TCP/IP-Header entfernt, das ICMP-Paket aufs TUN-Device geschrieben und durch das Betriebssystem an das Programm *ping* weitergeleitet.

ping misst die Zeit, bis der *echo reply* eingetroffen ist und gibt die Zeit aus. Nun beginnt das Spiel von Neuem.

Auf diese Weise hat das Programm *ping* auf Peer 1 ein System mit lokaler IP-Adresse (10.0.0.5) angesprochen und eine Antwort erhalten, obwohl dieses System nicht über ein lokales Netzwerk verbunden ist.

5. Implementierung

Das zugrunde liegende Konzept der VPN-Software wurde im vorherigen Kapitel erläutert. Nun soll gezeigt werden, wie einzelne Aspekte dieses Konzepts konkret implementiert wurden.

5.1 Betriebssystem und Programmiersprache

Auch wenn im Konzept weitestgehend plattformunabhängige Lösungen für die einzelnen Teilprobleme vorgeschlagen wurden (Sockets beispielsweise werden von jedem Betriebssystem unterstützt), muss ein Betriebssystem als primäre Entwicklungsumgebung gewählt werden. Portierungen auf andere Betriebssysteme können zu einem späteren Zeitpunkt vollzogen werden.
Durch die Quellenoffenheit und die weitestgehend standardisierten Schnittstellen fiel die Wahl auf das Betriebssystem Linux (Ubuntu Distribution 8.04[1]). Die meisten verfügbaren (und alle in dieser Arbeit verwendeten) Entwicklungsbibliotheken sind quelloffen und kostenlos verfügbar. Die Handhabung von Schnittstellen zur Netzwerkprogrammierung ist gut dokumentiert und gestaltet sich durch die Verwendung von Sockets, wie bereits erläutert, besonders einfach.

Da die meisten Bibliotheken und Schnittstellen für Linux am einfachsten über die Programmiersprache C angesprochen werden können, ist auch die VPN-Software in dieser Arbeit in C implementiert.

5.2 Basisimplementierung

Die private IP-Adresse, die der Peer im VPN haben soll, wird der Einfachheit halber als Parameter bei Programmstart mit angegeben. Ein weiterer Parameter gibt an, ob der Peer einem vorhandenes VPN beitreten möchte oder ob er selbst ein VPN eröffnet und damit zum Bootstrap-Peer wird, über den andere Peers dem Netzwerk beitreten können.

[1] http://www.ubuntu.com

5.3 Chimera

Chimera[2] ist ein strukturiertes Peer-to-Peer-Overlay, welches die im Konzept beschriebenen Anforderungen am besten erfüllt.
Die aktuelle Version 1.20 für Linux ist in C programmiert und wurde an der *University of California, Santa Barbara* entwickelt. Es basiert auf den strukturierten Peer-to-Peer-Overlays *Pastry* und *Tapestry*, die erstmals 2001 vorgestellt wurden, jedoch in Java geschrieben sind.

Die Vorteile von Chimera gegenüber anderen Peer-to-Peer-Overlays liegen in der Kompaktheit und Ausgereiftheit. Es ist relativ klein und übersichtlich und implementiert ein einfaches Nachrichtensystem auf Basis des key-based-routing-Mechanismus.

Die Keys in Chimera sind MD5-Hashwerte. In unserem Fall handelt es sich um die Hashwerte privater IP-Adressen. Möchte ein Peer beispielsweise die öffentliche IP-Adresse von 10.0.0.5 erfragen, sendet er die Nachricht an den Peer, der für den Key zuständig ist, der mit der Funktion key_makehash(10.0.0.5) erzeugt wird.

Chimera ermöglicht die Registrierung eigener Nachrichtentypen. Auf jeden dieser Nachrichtentypen kann auf unterschiedliche Weise reagiert werden. Für unsere Zwecke genügen drei Nachrichtentypen: REQUEST_PUBLIC_IP, RESPOND_PUBLIC_IP und RESPOND_NOT_AVAILABLE

Der erste Nachrichtentyp dient dazu eine Anfrage an einen beliebigen Peer zu stellen, um dessen öffentliche IP-Adresse in Erfahrung zu bringen. Nachrichten vom zweiten Typ transportieren die Antwort, also die öffentliche IP-Adresse, zurück zu dem Peer, der die Anfrage gestellt hat. Der dritte Nachrichtentyp wird dann als Antwort gesendet, wenn es keinen Peer mit der entsprechenden privaten IP-Adresse im VPN gibt.
Wie bereits erläutert wurde, wird der gesamte Keyraum abgedeckt, sodass die Nachricht REQUEST_PUBLIC_IP definitiv bei irgendeinem Peer ankommt und dann gegebenenfalls mit RESPOND_NOT_AVAILABLE von diesem Peer beantwortet werden kann.

Wurden die Nachrichtentypen registriert, tritt der Peer mit dem Hashwert seiner privaten IP-Adresse als Key dem Peer-to-Peer-Overlay bei. Über diesen Key kann er nun angesprochen werden. Die Verteilung des restlichen Keyraums wird intern von Chimera übernommen und spielt keine größere Rolle für die Funktionsweise des VPN.

5.4 TUN-Device

Der TUN-Kerneltreiber ist bereits in die Ubuntu-Distribution integriert. Das TUN-Device befindet sich unter /dev/net/tun und kann ganz einfach mit open() geöffnet werden. Es gibt die Betriebsmodi *blocking* und *non-blocking*. Aufgrund der einfacheren Handhabung wird das TUN-Device im (Standard-)Betriebsmodus *blocking* geöffnet. Der Filedeskriptor, den open() zurückliefert, kann nun für die Lese- und Schreiboperationen verwendet werden.

5.5 TCP

Obwohl die Performance bei der Datenübertragung eine wichtige Rolle spielt und mit UDP in dieser Hinsicht aufgrund der fehlenden Antworten (ACKs) ein besseres Ergebnis erzielt werden kann, fällt die Wahl des Transportprotokolls auf TCP.

[2] http://current.cs.ucsb.edu/projects/chimera/

Sowohl für den Verbindungsaufbau, als auch für die spätere Datenübertragung, empfiehlt sich aufgrund der Verwendung von SSL zur Verschlüsselung der Einsatz eines zuverlässigen und verbindungsorientierten Protokolls wie TCP. Das Konzept der Tunnel im VPN wird durch den Einsatz von TCP besser repräsentiert, als es mit dem Einsatz von UDP repräsentiert werden könnte.

Allerdings gibt es auch für UDP Möglichkeiten zur Verschlüsselung mit DTLS (Datagram Transport Layer Security), sodass eine zukünftige Untersuchung durchaus sinnvoll sein kann.

5.6 Tunneling

Wie bereits erläutert wurde, gibt es beim Verbindungsaufbau aus Sicht des TCP-Protokolls immer einen Server und einen Client. Da jeder Peer dazu in der Lage sein muss sowohl Verbindungen auf seinen Wunsch hin aufzubauen, als auch Gesuche zum Aufbau einer Verbindung von anderen Peers zu akzeptieren, muss jeder Peer Server und Client zugleich sein.

In der Rolle des Servers bindet der Peer einen festen Port an einen Socket und lauscht an diesem Socket auf eingehende Verbindungen. Dies wird erreicht über die Funktionen bind() und listen(), wie sie in jedem Lehrbuch zur Netzwerkprogrammierung beschrieben sind. Der listen-Socket wird stets überwacht. Kommt ein Verbindungsgesuch herein, wird versucht mit accept() eine Verbindung herzustellen. Ein neuer Filedeskriptor wird angelegt, der für das Lesen und Schreiben des Datenverkehrs verwendet werden kann.

In der Rolle des Clients genügt der Aufruf von connect(), um eine Verbindung zu einem Server aufzubauen. Auch hier wird ein neuer Filedeskriptor angelegt.

Nun müssen verschiedene Vorgänge parallel ablaufen. Zum einen muss der listen-Socket auf eingehende Verbindungen überprüft werden. Zum anderen muss überprüft werden, ob am TUN-Device Pakete anliegen, die entweder auf den korrekten Tunnel geroutet werden oder dazu dienen einen neuen Tunnel zu öffnen. Des Weiteren müssen auch noch sämtliche geöffnete Tunnel (Sockets) auf hereinkommende Pakete überprüft werden, die an das TUN-Device weitergeleitet werden.

Da sowohl das TUN-Device, als auch sämtliche Sockets im *blocking-mode* laufen, bietet sich der Einsatz der Funktion select() an. select() überprüft, ob an einem Satz von Filedeskriptoren (FD_SET) eine Änderung anliegt. Wie die Änderung genau aussieht und wie darauf reagiert werden soll, kommt auf den jeweiligen Filedeskriptor an. Liegt eine Änderung am listen-Socket an, wird darauf mit accept() reagiert. Liegt eine Änderung am TUN-Device an, wird entweder das Paket weitergeleitet oder ein neuer Tunnel geöffnet (write() oder connect()). Liegt eine Änderung an einem der geöffneten Tunnel an, werden die Daten mit read() ausgelesen und auf das TUN-Device geschrieben.

select() hilft dabei Parallelität zu erzeugen ohne den Prozessor unnötig durch aktives Warten in einer Endlosschleife zu belasten. Der Prozess wird schlafen gelegt und durch das Betriebssystem wieder aufgeweckt, sobald Änderungen an den Filedeskriptoren anliegen.

5.7 Routing

Das Routing der Pakete, die vom TUN-Device gelesen werden, auf die richtigen Tunnel funktioniert so, wie im Konzept beschrieben und bedarf keiner weiteren Erläuterung.

5.8 SSL

SSL ist ein standardisiertes Verschlüsselungsprotokoll. OpenSSL[3] ist eine konkrete Implementierung dieses Standards, aktuell in Version 0.9.8g. OpenSSL ist OpenSource und ausgereift, weswegen es im Rahmen dieser Studienarbeit eingesetzt wird.

Die Hauptkomponenten und Authentifizierungsmöglichkeiten von SSL wurden bereits vorgestellt. In einer klassischen Client-/Server-Architektur ist es durchaus denkbar, dass sich nur der Server gegenüber dem Client authentifiziert, wie es in nahezu jeder Webanwendung, die auf HTTPS setzt, der Fall ist. Die VPN-Software erfordert jedoch eine beidseitige Authentifizierung über ein Zertifikat, denn beide Peers sind gleichberechtigt. Bei der Initialisierung von SSL wird dies durch die Angabe der beiden Flags SSL_VERIFY_PEER und SSL_VERIFY_FAIL_IF_NO_PEER_CERT erreicht.

Der SSL-Handshake muss ohne das Auftreten eines Fehlers erfolgreich durchgeführt werden, bevor der Tunnel zwischen den Peers aufgebaut werden kann. Tritt ein Fehler auf, wird der Versuch den Tunnel aufzubauen abgebrochen.

Anmerkung: Streng genommen verwendet die VPN-Software für die Lese- und Schreiboperationen Funktionen, die OpenSSL anbietet. Der Datenverkehr muss ver- und entschlüsselt werden. Die Funktionen sind analog zu read() und write() und heißen SSL_read() bzw. SSL_write(). Zum besseren Verständnis ist in dieser Dokumentation immer nur die Rede von den Standardfunktionen und nicht von deren SSL-Äquivalenten.

5.9 Verbesserungsmöglichkeiten

Der Fokus der Implementierung liegt in der Erzeugung des VPNs und dem Austausch von Daten. Verbesserungswürdig ist die Wahrung der virtuellen Netzwerkstruktur durch bessere Datenstrukturen, beispielsweise die Verwendung verketteter Listen an Stelle fester Arrays. Chimera bietet aktuell keine Möglichkeit eine Nachricht zu verschicken, die dem Overlay mitteilt, dass ein Peer das Netzwerk verlassen möchte. Da das Verlassen des Netzwerks für die Untersuchung hinsichtlich der Vermeidung eines Flaschenhalses nicht weiter relevant ist, wurde auf dessen Implementierung verzichtet.

Eine VPN-Software im Produktiveinsatz muss in der Lage sein mit Firewalls und NAT (Network Address Translation) umgehen zu können. Es gibt verschiedene Techniken zum NAT- bzw. Firewall-Traversal, beispielsweise das sogenannte *Hole Punching*.

Der Einfachheit halber wird das VPN über einen Bootstrap-Peer erzeugt, der als einziger Einstiegspunkt für andere Peers dient. Ideal wäre ein Szenario, in dem ein Beitrittskandidat lediglich einen Teilnehmer des Netzwerks kennt und über diesen dem Netzwerk beitreten kann und nicht zwangsläufig den Bootstrap-Peer kennen muss. Verteilte Einstiegspunkte erhöhen die Flexibilität des Netzwerks.

Wird ein TCP-Paket mit der maximalen MTU (*Maximum Transmission Unit*) von einer Legacy-Anwendung erzeugt (in einem Ethernet beträgt die größtmögliche MTU 1500 Bytes), führt das Verpacken dieses Pakets in ein TCP-Paket durch die VPN-Software unweigerlich zu dessen Fragmentierung, weil ein weiterer TCP/IP-Header hinzugefügt wird, der einige Bytes Platz benötigt. Dadurch kann die Performance eingeschränkt werden. Um dies zu vermeiden, muss die Legacy-Anwendung dazu gebracht werden kleinere Pakete zu erzeugen, damit die 1500 Bytes nicht beim Verpacken des Pakets überschritten werden.

[3] http://www.openssl.org

Im Bereich Sicherheit sind weitere Überlegungen notwendig in Bezug auf die Ablaufzeit von Zertifikaten, um Angriffen in einem Netzwerk vorzubeugen, das über einen längeren Zeitraum besteht. Es könnte durchaus Sinn machen nach einiger Zeit einen *SSL-Rehandshake* durchzuführen, der beide Peers aufs Neue authentifiziert.

6. Evaluation

Das bereits vorgestellte OpenVPN wurde als Testprogramm für einen direkten Vergleich mit der Implementierung dieser Studienarbeit herangezogen. Mit Hilfe zweier Testaufbauten sollte evaluiert werden, inwiefern der Server von OpenVPN die verfügbaren Netzwerkressourcen einschränkt und einen Single Point of Failure darstellt und ob das hier vorgestellte Konzept eines Peer-to-Peer-basierten VPNs dazu in der Lage ist diesem Problem auf effiziente Weise zu begegnen.

Zur Evaluation wurde der Netzwerkverkehr sowohl auf dem OpenVPN-Server im OpenVPN-Testaufbau als auch auf dem Bootstrap-Peer im Peer-to-Peer-Testaufbau beobachtet. Mit Hilfe des Programms *ping* wurden Datenpakete zwischen den einzelnen Peers ausgetauscht. Aus dem Verhalten des Servers beziehungsweise des Bootstrap-Peers wurden dann einige theoretische Überlegungen abgeleitet, die am Ende dieses Kapitels vorgestellt werden.

6.1 Evaluationsumgebung

Der OpenVPN-Testaufbau sah einen Server und zwei Clients vor. Damit sich die Clients untereinander „sehen“ können, muss in der Serverkonfigurationsdatei die Zeile

```
client-to-client
```

hinzugefügt werden. Standardmäßig sieht die Konfiguration nicht vor, dass sich die Clients untereinander sehen können. Der Server hat die virtuelle IP-Adresse 10.8.0.1, Client A und B haben 10.8.0.5 und 10.8.0.9. Auf allen drei Rechnern läuft eine Standardinstallation des Betriebssystems Ubuntu in Version 8.04. Für OpenVPN benötigte Pakete wurden nachinstalliert (beispielsweise OpenSSL). Zur Beobachtung des Netzwerkverkehrs auf dem Server wurde Wireshark 1.0.2[1] eingesetzt.

Der Peer-to-Peer-Testaufbau bestand aus einem Bootstrap-Peer (Peer A) mit der virtuellen IP-Adresse 10.0.0.1 und drei Peers (Peer B, C und D) mit den virtuellen IP-Adressen 10.0.0.2, 10.0.0.3 und 10.0.0.4. Auf allen vier Systemen war ebenfalls Ubuntu 8.04 installiert und zur Beobachtung des Netzwerkverkehrs auf dem Bootstrap-Peer wurde, wie im OpenVPN-Testaufbau, Wireshark eingesetzt.

[1] http://www.wireshark.org - Wireshark ist die Weiterentwicklung von Ethereal unter neuem Namen.

6.2 Testbeschreibungen

Folgende Tests wurden im OpenVPN-Setup durchgeführt:

1. Nach der Initialisierung des OpenVPN-Servers wird Wireshark gestartet. Nun tritt Client A und anschließend Client B bei. Beim Verbindungsaufbau soll das SSL-Handshake auf dem Server aufgezeichnet werden.

2. Auf Client A wird der Befehl

```
> ping 10.8.0.9 -c 20 -s 1024
```

aufgerufen, um zu überprüfen, ob der Datenverkehr von Client A nach Client B über den Server läuft.

Die Tests im Peer-to-Peer-Setup sahen folgendermaßen aus:

1. Peer B sendet einen Ping zum Bootstrap-Peer A, um den SSL-Handshake zu protokollieren, durch Aufruf des Befehls

```
> ping 10.0.0.1 -c 3 -s 1024
```

2. Peer B sendet einen Ping zu Peer C. Auf Peer A wird der Netzwerkverkehr beobachtet, um zu überprüfen, ob Peer A von der Kommunikation über den direkten Tunnel zwischen B und C etwas mitbekommt. Der Aufruf auf dem System von Peer B lautet:

```
> ping 10.0.0.3 -s 1024
```

3. Der ping-Befehl aus Test 2 wird laufen gelassen. Peer A sendet nun ebenfalls einen Ping zu Peer C:

```
> ping 10.0.0.3
```

Anschließend sendet Peer D einen Ping zu Peer A:

```
> ping 10.0.0.1 -s 1024
```

Es soll damit überprüft werden, wie Peer A mit zwei geöffneten Tunneln umgeht.

6.3 Testergebnisse und Bewertung

Betrachtet man nach der Initialisierung des OpenVPN-Servers den Netzwerkverkehr beim Beitritt eines Clients, fällt die erste Gemeinsamkeit beider Programme auf. Der SSL-Handshake wird durchgeführt und nimmt einige Millisekunden in Anspruch, was sich nicht vermeiden lässt. Da der Zertifikatsaustausch und die Einigung auf einen Verschlüsselungsalgorithmus nur einmalig pro Verbindungsaufbau erfolgen muss, fällt der Rechenaufwand und die Verzögerung kaum ins Gewicht.

Die Durchführung von Test 2 zeigt, dass der Datenverkehr von Client A nach Client B in der Tat über den Server abgewickelt wird. Eine direkte Verbindung wurde nicht aufgebaut. Außerdem kann festgestellt werden, dass OpenVPN in seiner Standardkonfiguration auf UDP setzt. Alle 10 Sekunden wird ein Paket verschickt, das von den Clients beantwortet werden muss (ähnlich einem Ping, genannt *keepalive*). UDP ist ein zustandsloses Protokoll.

Auf diese Weise wird sichergestellt, dass die Clients noch erreichbar sind und tote Verbindungen zu unerreichbaren Clients auf Serverseite beendet werden, um dafür reservierte Ressourcen wieder freizugeben.

Die Beobachtung des Netzwerkverkehrs auf Peer A bei Durchführung des ersten Peer-to-Peer-Tests zeigt ebenfalls den SSL-Handshake, der vom Umfang her dem SSL-Handshake im OpenVPN-Setup entspricht. Obwohl jeder einzelne SSL-Handshake gleich viel Last erzeugt, wird im Peer-to-Peer-basierten VPN durch die Verwendung mehrerer Tunnel, also mehrerer paralleler SSL-Verbindungen, effektiv etwas mehr Last erzeugt.

Test 2 hingegen offenbart den Unterschied zwischen beiden VPN-Lösungen. Peer A bekommt von der Kommunkation zwischen Peer B und C nichts mit.

Der dritte Test zeigt, dass Peer A seinen Upstream effizient nutzen kann. Die Kommunikation zwischen Peer B und C belastet ihn gar nicht. Lediglich die eigene Kommunikation mit Peer C und anschließend die parallele Kommunikation mit Peer D belastet den Upstream. Dabei wird für die Kommunikation mit Peer D durch die erhöhte Paketgröße etwas mehr Bandbreite in Anspruch genommen.

Aus den Testergebnissen lässt sich nun die Überlegung ableiten, dass die Schwachstelle der Client-/Server-Architektur insbesondere dann eine Rolle spielt, wenn der zentrale Server eine Vielzahl von Clients bedienen muss. Erzeugen diese Clients parallel einen hohen Datenverkehr und muss der Server alle Pakete weiterleiten, wird das Netzwerk durch ihn ausgebremst. Es besteht die Gefahr, dass der Server unter der Last hereinkommender Datenpakete zusammenbricht und vollständig ausfällt, sodass gar keine Daten mehr fließen können. Das Problem des Single Point of Failure (also der Totalausfall und das Versiegen des gesamten Datenverkehrs) macht sich somit nicht nur bei einem gezielten Angriff auf das System bemerkbar, sondern kann auch in Folge von starkem (gewünschten) Datenverkehr auftreten.

Wenn der Datenverkehr hingegen relativ gering ausfällt (ping erzeugt nur sehr wenig Traffic), kann auch ein zentraler Server die Pakete von einer Vielzahl von Clients weiterleiten, ohne, dass das gesamte VPN darunter leidet.

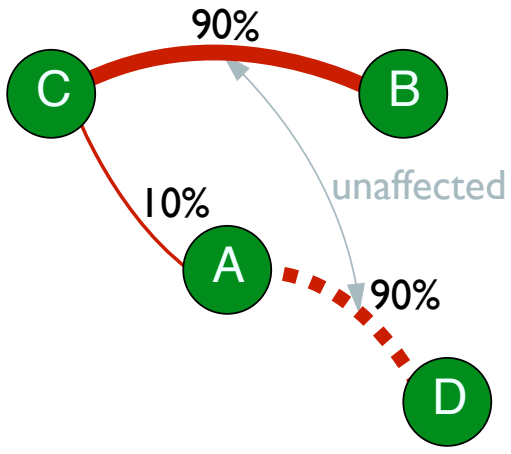

Abbildung 6.1: „Nutzung des Upstreams“

Test 2 und 3 im Peer-to-Peer-Setup verdeutlichen, dass das Peer-to-Peer-basierte VPN eine effizientere Herangehensweise an das Problem des Flaschenhalses darstellt. Es werden nur die Netzwerkkapazitäten der Peers ausgelastet, die auch an der Kommunikation beteiligt

sind. Peer A wird durch die Kommunikation zwischen Peer B und C nicht gebremst. Dadurch sind auch VPNs mit einer Vielzahl von Peers denkbar, die allesamt parallel große Mengen von Daten versenden. Auch für kleine Datenmengen entsteht durch die Peer-to-Peer-Architektur kein Nachteil. Lediglich das Nachrichtensystem von Chimera erzeugt einen geringen Kommunikationsoverhead.

Natürlich kann auch im Peer-to-Peer-basierten VPN ein einzelner Peer ausfallen, wenn er mehr Anfragen von anderen Peers bekommt, als er zu verwalten im Stande ist. Dieses Problem existiert allerdings in jeder Netzwerktopologie und muss ganz anders gelöst werden (beispielsweise durch Spiegelung des Systems), ist aber im Rahmen dieser Arbeit nicht Gegenstand der Untersuchung.

Des Weiteren ist im Peer-to-Peer-basierten VPN wichtig, dass die Informationen, die auf den einzelnen Peers gespeichert werden, korrekt synchronisiert werden. Ein unsachgemäßes Verlassen des Netzwerks durch einen Peer führt zu Datenverlust. Der Datenverlust muss vermieden werden, beispielsweise durch redundante Speicherung der Daten.

7. Zusammenfassung und Ausblick

Obwohl zunächst verschiedene Schwächen einer klassischen VPN-Software auf Basis einer Client-/Server-Architektur erörtert wurden, spielt eine Schwäche eine besondere Rolle: Die zunehmende Gefahr, dass die Performance beim Datenverkehr durch höher werdende Upstreamraten der Clients immer schlechter wird und der Server das gesamte Netzwerk ausbremst oder gar vollständig ausfällt durch die Menge der gleichzeitig hereinkommenden Daten.

Die Anwendungsbereiche von VPNs sind sehr vielfältig, doch in einer Firmenumgebung und selbst bei Computerspielen ist der Datenverkehr nach wie vor verhältnismäßig gering. Konstante Durchsatzraten werden vermehrt bei der Übertragung vieler bzw. großer Dateien erreicht, beispielsweise unter Verwendung von Dateiaustauschsprotokollen wie SMB oder CIFS[1]. Eine Orientierung weg von der klassischen Client-/Server-Architektur hin zur Peer-to-Peer-Technologie ist insbesondere im Bereich des Dateiaustauschs erforderlich. Das BitTorrent-Protokoll[2] verwendet bereits Peer-to-Peer-Technologien.

Ein Peer-to-Peer-gestütztes VPN weist eine große Flexibilität und Mobilität auf, die durch den geringen Konfigurationsaufwand und die Gleichberechtigung aller beteiligten Systeme erzielt wird. Es wird keine zusätzliche Infrastruktur (ein Server) benötigt. Dies macht das Peer-to-Peer-gestützte VPN insbesondere für Privatanwender, die ein verhältnismäßig kleines Netzwerk aufbauen möchten, zu einer echten Alternative, beispielsweise, um spontan ein Netzwerk für Computerspiele zu eröffnen.

Im Firmenbereich bildet Sicherheit die oberste Priorität. Die Performance bei der Datenübertragung muss nicht immer optimal sein, solange die Daten vor dem Zugriff durch Dritte gesichert sind. Für den Einsatz in Firmennetzwerken bildet das Peer-to-Peer-gestützte VPN noch keine echte Alternative. Die Flexibilität und Mobilität durch die Gleichberechtigung der Teilnehmer ist nicht notwendig. Gerade die Privilegierung eines Servers gegenüber den Clients führt zu einer einfacheren Wahrung der Sicherheit, weil das Netzwerk weniger Angriffsfläche bietet. Oftmals wird gerade in Firmenumgebungen ein Server eingesetzt, um einen bestimmten Dienst für viele Clients anzubieten. Die Sichtbarkeit von Clients untereinander ist nicht unbedingt notwendig. In Fällen, in denen Clients auch untereinander kommunizieren können, könnte es sogar wünschenswert sein, dass an einer zentralen Stelle die Kommunikation überwacht wird, sei es aus Sicherheitsgründen, um Datendiebstahl

[1] http://www.samba.org/cifs/
[2] http://www.bittorrent.com/

innerhalb der Firma vorzubeugen oder um zu protokollieren, welcher Client über welche Daten verfügt.

Auf der anderen Seite steht die verbesserte Auslastung der Upstreamkapazitäten eines jeden Kommunikationsteilnehmers und eine Dezentralisierung könnte dabei helfen den Server als zentrales Angriffsziel zu entlasten, um die Stabilität des Netzwerks insbesondere bei einem *Denial-of-Service*-Angriff zu erhöhen. Zwar stellt auch im Peer-to-Peer-Bereich die Zertifizierungsstelle ein zentrales Element dar, jedoch kann diese auch offline betrieben werden, sodass diese kein unmittelbares Ziel mehr darstellt für einen Angriff.

Es muss also je nach Anwendungsgebiet unterschieden werden, ob lieber auf klassische VPN-Software gesetzt werden soll oder ob ein Peer-to-Peer-gestütztes VPN eine Alternative darstellt. Zusammenfassend lässt sich sagen, dass zum gegenwärtigen Zeitpunkt insbesondere der Privatanwender vom Einsatz eines Peer-to-Peer-gestütztes VPN aus oben genannten Gründen profitiert.

Abkürzungsverzeichnis

DHT	Distributed Hash Table
KBR	Key-Based-Routing
MTU	Maximum Transmission Unit
RTT	Round-Trip-Time
SSL	Secure Socket Layer
TLS	Transport Layer Security
TTL	Time-To-Live
VPN	Virtual Private Network

Literaturverzeichnis

[Boyd93] C. Boyd. Security architecture using formal methods. *IEEE Journal on Selected Topics in Communications* Band 11, 1993, S. 694–701.

[DiAl99] T. Dierks und C. Allen. RFC 2246: The TLS Protocol Version 1, Januar 1999. Status: PROPOSED STANDARD.

[KeAt98] S. Kent und R. Atkinson. RFC 2401: Security Architecture for the Internet Protocol, November 1998. Obsoletes RFC1825. Status: PROPOSED STANDARD.

[ScFe99] Bruce Schneier und Niels Ferguson. *A Cryptographic Evaluation of IPsec*. Counterpane Internet Security, Inc. Februar 1999.

[URLa] OpenSSL. http://www.openssl.org.

[URLb] OpenVPN. http://www.openvpn.net.

[URLc] socat. http://www.dest-unreach.org/socat/.

[URLd] VPN-X. http://birdssoft.com.

[URLe] X-Bone. http://www.isi.edu/xbone/.

[URLf] Hamachi. http://www.hamachi.cc.

[URLg] Ubuntu Distribution. http://www.ubuntu.com.

[URLh] Chimera. http://current.cs.ucsb.edu/projects/chimera/.

[URLi] Wireshark. http://www.wireshark.org.

[URLj] CIFS. http://www.samba.org/cifs/.

[URLk] BitTorrent. http://www.bittorrent.com.

[Wehr05] Steinmetz Wehrle. *Peer-to-Peer Systems and Applications*. Springer. 2005.